Two-dimensional Context-free Grammars

Daniel Průša*

Charles University, Department of Computer Science, Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic, e-mail: prusa@barbora.ms.mff.cuni.cz

Abstract. We present a generalization of context-free grammars to two-dimensions and define picture languages generated by these grammars. We examine some properties of the formed class and we describe how these languages can be recognized by two-dimensional forgetting automata.

1 Introduction

The goal of this paper is to present one of possible generalizations of concepts of context-free grammars and languages to two dimensions. Informally, a two-dimensional string (called a picture) is defined as a rectangular array of symbols from a finite alphabet. A picture language is a set of pictures.

Some proposals of two-dimensional context-free languages already exist ([4], [2]), however a complete theory has not been formed yet. It is a difficult task. The situation is rather complicated even in case of regular languages. We emphasize that our ambitions are not to claim what two-dimensional context-free languages should be. We generalize concepts of context-free grammars in a natural way only and study the formed class of picture languages. However, in the text, we use terms like two-dimensional context-free grammar, resp. language to refer to them.

Our generalized grammars have productions whose left sides are non-terminals and the right sides are matrixes of terminals and non-terminals. This idea is not original. It can be found for example in [7], where productions with the rigth side restricted to one row or column are considered only. The other example is in [8]. Some basic facts, that we extend, are mentioned there.

Our results on the class of context-free languages include the facts that not all languages recognized by (two-dimensional) finite state automata are context-free and that the restricted grammars from [7] are weaker then

* Supported by the Grant Agency of the Czech Republic, Grant-No. 157/1999/A INF/MFF

the presented grammars. In addition, we describe how context-free languages can be recognized by two-dimensional forgetting automata. This construction is based on results in [8] and [1].

2 Picture Languages

We assume that the reader is familiar with the theory of one-dimensional languages as can be found for example in [3]. We extend some basic definitions from the one-dimensional theory now. More details can be found in [4].

Definition 1. *A* picture *over a finite alphabet Σ is a two-dimensional rectangular array (matrix) of elements of Σ. Σ^{**} denotes the set of all pictures over Σ. A* picture language *over Σ is a subset of Σ^{**}.*

Let $O \in \Sigma^{**}$ be a picture. $rows(O)$, resp. $cols(O)$ denotes the number of rows, resp. columns of O. The pair $rows(O) \times cols(O)$ is called the *size* of O. We say that O is a square picture of the size n if $rows(O) = cols(O) = n$. The *empty picture* Λ is the only picture of the size 0×0. For integers i, j such that $1 \leq i \leq rows(O)$, $1 \leq j \leq cols(O)$, $O(i, j)$ denotes the symbol in O at the coordinate (i, j). A *sub-picture* of O is a sub-matrix of it.

We use $[a_{ij}]_{m,n}$ to denote the matrix

$$\begin{matrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{matrix}$$

We define two binary operations – the row and the column concatenation. Let $A = [a_{ij}]_{k,l}$ and $B = [b_{ij}]_{m,n}$ be non-empty pictures over Σ. The column concatenation $A \varobar B$ is defined iff $k = m$ and the row concatenation $A \ominus B$ iff $l = n$. The results of the operations are given by the following schemes:

$$A \varobar B = \begin{matrix} a_{11} & \dots & a_{1l} & b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{kl} & b_{m1} & \dots & b_{mn} \end{matrix} \qquad A \ominus B = \begin{matrix} a_{11} & \dots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{kl} \\ b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{matrix}$$

Moreover, the column and the row concatenation of A and Λ is always defined and Λ is the neutral element for both operations.

The unary operation $\bigoplus$ is defined on a set of matrixes whose elements are pictures over an alphabet. Let P_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$ be pictures over Σ such that $\forall i \in \{1, \ldots, m\}\, rows(P_{i1}) = rows(P_{i2}) = \ldots = rows(P_{in})$ and $\forall j \in \{1, \ldots, n\}\, cols(P_{1j}) = cols(P_{2j}) = \ldots = cols(P_{mj})$. $\bigoplus[P_{ij}]_{m,n}$ is defined as $P_1 \ominus P_2 \ominus \ldots \ominus P_m$, where $P_k = P_{k1} \oplus P_{k2} \oplus \ldots \oplus P_{kn}$.

In our descriptions we use the system of coordinates in a picture depicted in Fig. 1. Speaking about a position of a specific field, we use words like up, down, right, left, first row, last row etc. with respect to this scheme.

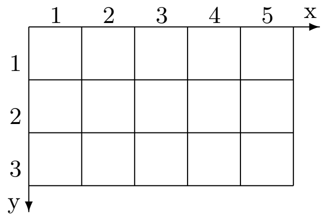

Fig. 1. The system of coordinates used in our picture descriptions.

3 Two-dimensional Automata

The two-dimensional Turing machine works on a two-dimensional tape. It can move its head left, right, up and down. We give its formal definition only. Terms like configuration, computation, accepting, recognized language, etc. are defined in a natural way. Details can be found in [4].

Definition 2. Two-dimensional Turing machine *is a tuple* $(Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$*, where* Q *is a finite set of states,* Σ *is a tape alphabet,* $\Sigma_0 \subset \Sigma$ *is an input alphabet,* $q_0 \in Q$ *is the initial state,* $Q_F \subseteq Q$ *is a set of final states and* $\delta : \Sigma \times Q \to 2^{\Sigma \times Q \times \mathcal{M}}$ *is a transition relation.* $\mathcal{M} = \{L, R, U, D, N\}$ *is the set of automaton movements (left, rigth, up, down, no movement). We always assume that there is a distinguished symbol* $\# \in \Sigma \setminus \Sigma_0$ *called the background symbol.*

A two-dimesional Turing machine is *bounded* iff the head does not leave an input during the computation (when it encounteres # it returns in the next step and does not rewrite this symbol). We consider bounded machines in the following text only. A two-dimensional finite state automaton is a two-dimensional Turing machine that does not rewrite any symbol during its computation. We abbreviate it as FSA, deterministic FSA as $DFSA$.

4 Two-dimensional Context-free Grammars

Definition 3. Two-dimensional context-free grammar G *is a tuple* $(V_N, V_T, S_0, \mathcal{P})$*, where* V_N *is a finite set of non-terminals,* V_T *is a finite set of terminals,* $S_0 \in V_N$ *is the initial non-terminal and* $\mathcal{P}$ *is a finite set of productions of the form* $N \to W$*, where* $N \in V_N$ *and* $W \in (V_N \cup V_T)^{**} \setminus \{\Lambda\}$*. In addition,* $\mathcal{P}$ *can contain* $S_0 \to \Lambda$*. In such a case,* S_0 *is not a part of any right side of productions.*

Definition 4. *Let* $G = (V_N, V_T, S_0, \mathcal{P})$ *be a planar context-free grammar. We define a picture language* $\mathcal{L}(G, N)$ *over* V_T *for every* $N \in V_N$*. The definition is given by the following recursive description:*

A) If $N \to W$ *is a production in* $\mathcal{P}$ *and* $W \in {V_T}^{**}$*, then* W *is in* $\mathcal{L}(G, N)$*.*
B) Let $N \to [A_{ij}]_{m,n}$ *be a production in* $\mathcal{P}$ *(not* $S_0 \to \Lambda$*) and* P_{ij} *(*$i = 1, \ldots, n$ $j = 1, \ldots, m$*) pictures such that: For every pair of indexes* i*,*j*, if* A_{ij} *is a terminal then* P_{ij} *is the picture of the size* 1×1 *whose the only field contains the symbol* A_{ij}*. If* A_{ij} *is non-terminal then* $P_{ij} \in \mathcal{L}(G, A_{ij})$*. In addition,* $\bigoplus[P_{ij}]_{m,n}$ *is defined. Then* $\bigoplus[P_{ij}]_{m,n}$ *is an element of* $\mathcal{L}(G, N)$*.*

The set $\mathcal{L}(G, N)$ *contains just all pictures that can be obtained by applying a finite sequence of rules A) and B). The language* $\mathcal{L}(G)$ *generated by the grammar* G *is defined as the language* $\mathcal{L}(G, S_0)$*.*

We abbreviate a two-dimensional context-free grammar as CFG. $\mathcal{L}(CFG)$ is the class of all two-dimensional context-free languages. CF stands for context-free. An equivalent definition of a language generated by a context-free grammar is based on a generalization of derivation trees.

Definition 5. *Let* $G = (V_N, V_T, S_0, \mathcal{P})$ *be a* CFG*. A derivation tree for* G *is every tree* T *satisfying:*

- T *has at least two vertices.*
- *Each vertice* v *of* T *is labeled by a pair* $(a, k \times l)$*. If* v *is a leaf then* $a \in V_T$*,* $k = l = 1$ *else* $a \in V_N$*,* $k, l \geq 1$ *are integers.*

– *Edges are labeled by pairs* (i,j). *Let us denote the set of labels of all edges connecting* v *with its descendant as* $I(v)$. *It holds that* $I(v) = \{1,\ldots,m\} \times \{1,\ldots,n\}$ *and* $m.n$ *is the number of descendants of* v.
– *Let* v *be a vertice of* T *labeled* $(N, k \times l)$, *where* $I(v) = \{1,\ldots,m\} \times \{1,\ldots,n\}$. *Let the edge labeled* (i,j) *connect* v *and its descendant* v_{ij} *labeled* $(A_{ij}, k_i \times l_j)$. *Then* $\sum_{i=1}^{m} k_i = k$, $\sum_{j=1}^{n} l_j = l$ *and* $N \to [A_{ij}]_{m,n}$ *is a production in* $\mathcal{P}$.

If $S_0 \to \Lambda \in \mathcal{P}$ *then the tree* T_Λ *with two vertices – the root labeled* $(S_0, 0 \times 0)$ *and the leaf labeled* $(\Lambda, 0 \times 0)$ *is a derivation tree for* G *too.*

Let T be a derivation tree for a CF grammar $G = (V_N, V_T, S, \mathcal{P})$, V set of its vertices. We assign a picture to each vertice of T by defining a function $p : V \to {V_T}^{**}$: if $v \in V$ is a leaf labeled $(a, 1 \times 1)$ then $p(v) = a$ else $p(v) = \bigoplus [P_{ij}]_{m,n}$, where $I(v) = \{1,\ldots,m\} \times \{1,\ldots,n\}$, $P_{ij} = p(v_{ij})$, v_{ij} is a descendant of v connected by the edge labeled (i,j). $p(T)$ is defined as $p(r)$, where r is the root of T. $p(T_\Lambda) = \Lambda$. Observation: if $v \in V$ is labeled $(N, k \times l)$ then $rows(p(v)) = k$, $cols(p(v)) = l$.

Lemma 1. *Let* $G = (V_N, V_T, S, \mathcal{P})$ *be a CF grammar and* $N \in V_N$.

1. *Let* T *be a derivation tree for* G *having its root labeled* $(N, k \times l)$. *Then* $p(T) \in \mathcal{L}(G, N)$.
2. *Let* O *be a picture in* $\mathcal{L}(G, N)$. *There exists a derivation tree for* G *with root labeled* $(N, k \times l)$ *such that* $rows(O) = k$, $cols(O) = l$ *and* $p(T) = O$.

Proof. The lemma follows directly from the previous definitions. □

Example 1. Let us define the picture language L over $\Sigma = \{a, b\}$.

$$L = \{O \mid O \in \{a,b\}^{**} \land \exists i, j \in \mathbf{N} : \ 1 < i < rows(O) \ \land \ 1 < j < cols(O) \ \land$$
$$\forall x \in \{1,\ldots,rows(O)\}, y \in \{1,\ldots,cols(O)\} : \ O(x,y) = a \Leftrightarrow x \neq i \ \land \ y \neq j\}$$

L is context-free. It is generated by the CF grammar $G = (V_N, \Sigma, S, \mathcal{P}, S)$, where $V_N = \{S, A, V, H, M\}$ and the set $\mathcal{P}$ consists of the following productions:

$$S \to \begin{matrix} A & V & A \\ H & b & H \\ A & V & A \end{matrix} \qquad A \to M \qquad A \to A\ M \qquad M \to a \qquad M \to \begin{matrix} a \\ M \end{matrix}$$

$$V \to b \qquad V \to \begin{matrix} b \\ V \end{matrix} \qquad H \to b \qquad H \to b\ H$$

The non-terminal A generates the language $\{a\}^{**} \setminus \{\Lambda\}$, M generates one-column pictures of a's, V generates one-column pictures of b's and finally H generates one-row pictures of b's.

Let us consider CF grammars with productions of the form $N \to a$, $N \to [A_{1j}]_{1,2}$ and $N \to [A_{i1}]_{2,1}$, where a is a terminal and A_{ij} are non-terminals. These grammars are presented in [7]. Let us denote them as $CFG2$. We proof that their generative power is less than the generative power of CFG's.

Theorem 1. *$\mathcal{L}(CFG2)$ is a proper subset of $\mathcal{L}(CFG)$.*

Proof. By a contradiction. Let $G = (V_N, V_T, S, \mathcal{P})$ be a $CFG2$ generating the language L from the Example 1. Let us consider an integer $n \geq 3$. We denote the set of all square pictures of the size n in L as L_1. n can be chosen sufficiently large so that no picture in L_1 equals to the right side of anyone production in $\mathcal{P}$. L_1 consists of $(n-2)^2$ pictures. At least $\lceil \frac{(n-2)^2}{|\mathcal{P}|} \rceil$ pictures are derived in the last step using the same production. Without loss of generality, let it be the production $S \to AB$. If n is sufficiently large there exist two pictures with different indexes of the row of b's (maximally $n-2$ pictures in L_1 can have the same index of the row of b's). Let us denote these pictures as O and $\overline{O}$. It holds $O = O_1 \varobar O_2$, $\overline{O} = \overline{O}_1 \varobar \overline{O}_2$, where $O_1, \overline{O}_1 \in \mathcal{L}(G, A)$ and $O_2, \overline{O}_2 \in \mathcal{L}(G, B)$. It implies $O = O_1 \varobar \overline{O}_2 \in \mathcal{L}(G)$. It is a contradiction, O contains b in the first and in the last column, but these b's are not in the same row. $\square$

Example 2. Let us define the language L over the alphabet $\Sigma = \{0, 1, x\}$ consisting just of all pictures $O \in \Sigma^{**}$ satisfying: 1) O is a square picture of an odd size, 2) $O(i,j) = x \Leftrightarrow i, j$ are odd indexes, 3) if $O(i,j) = 1$ then the i-th row or the j-th column (at least one of them) consists of 1's

Lemma 2. *L can be recognized by a $DFSA$.*

Proof. $DFSA$ automaton T recognizing L can be constructed as follows. T checks if an input picture is a square picture of an odd size. It can be done moving the head diagonally. The computation continues by scanning row by row and checking if symbols x are just in all fields with both indexes odd, in case of other fields containing and for the other positions the symbol 1 if the field and its four neighbours form one of the possible configurations as follows:

$$\begin{array}{ccc} & 1 & \\ x & 1 & x \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ 1 & 1 & 1 \\ & x & \end{array} \quad \begin{array}{ccc} & 1 & \\ 0 & 1 & 0 \\ & 1 & \end{array} \quad \begin{array}{ccc} & 0 & \\ 1 & 1 & 1 \\ & 0 & \end{array} \quad \begin{array}{ccc} & 1 & \\ 1 & 1 & 1 \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ \# & 1 & 1 \\ & x & \end{array} \quad \begin{array}{ccc} & \# & \\ x & 1 & x \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ 1 & 1 & \# \\ & x & \end{array} \quad \begin{array}{ccc} & 1 & \\ x & 1 & x \\ & \# & \end{array}$$

□

Theorem 2. *$\mathcal{L}(DFSA)$ is not a subset of $\mathcal{L}(CFG)$.*

Proof. Let $G = (V_N, V_T, S, \mathcal{P})$ be a CFG such that $\mathcal{L}(G) = L$, where L is the language from the Example 2. Without loss of generality, $\mathcal{P}$ does not contain any production of the form $A \to B$, where A, B are non-terminals. We take an odd integer $n = 2.k + 1$. Let L_1 be the set of all pictures in L of the size n. n is chosen sufficiently large so that no picture in L_1 equals to the right side of anyone production. We have $|L_1| = 2^k.2^k = 2^{n-1}$ (there is k columns and k rows, for each we can choose if the row, resp. colunm cosists of 1's or not). There is at least $\frac{2^{n-1}}{|P|}$ pictures in L_1 that are derived in the last step using the same production. Let it be the production $S \to [A_{ij}]_{p,q}$. Let the set of the given pictures be L_2. Without loss of generality, we assume that $p \geq q$. In addition, $p \geq 2$ (otherwise the production is of the form $A \to B$).

The goal is to show that there are two pictures $U, V \in L_2$ such that $U = \bigoplus[U_{ij}]_{p,q}$, $V = \bigoplus[V_{ij}]_{p,q}$, $U_{ij}, V_{ij} \in \mathcal{L}(G, A_{ij})$ (property (1)) and next that the first row of U does not equals to the first row of V (property (2) – in other words, it means U and V differs in one of columns with respect to the symbols 1). The number of all possible sequences

$$cols(U_{1,1}), cols(U_{1,2}), \ldots, cols(U_{1q}), rows(U_{1,1}), rows(U_{2,1}), \ldots, rows(U_{p1})$$

is bounded by n^{p+q}. There exists a set $L_3 \subseteq L_2$, $|L_3| \geq \frac{2^{n-1}}{|\mathcal{P}|.n^{p+q}}$ and each pair of pictures in L_3 has the property (1). L_2 contains a subset of $2^k = 2^{\frac{n-1}{2}}$ pictures, where each pair does not satisfy the property (2). It implies that the pair U, V exists in L_3 for some sufficiently large n. If we replace the sub-pictures $U_{1,1}, \ldots, U_{1q}$ in U by the sub-pictures $V_{1,1}, \ldots, V_{1q}$ (U_{1i} replaced by V_{1i}) we get the picture O that is in L again. But it is a contradiction, because O does not have all properties of pictures in L. □

5 Two-dimensional Forgetting Automata

Forgetting automata are bounded Turing machines that can rewrite the content of a field by the special symbol @ only (we say, they erase it). It is possible to characterize (one-dimensional) context-free languages using forgetting automata as it is shown in [5]. We extend some of these ideas – we show that two-dimensional context-free languages can be recognized by two-dimensional forgetting automata. The proof is strongly based on a technique of storing information in blocks that has been presented in [1], where relations between two-dimensional $NFSA$ and two-dimensional forgetting automata are studied.

Definition 6. Two-dimensional forgetting automaton *(*NFA*) is a two-dimensional bounded Turing machine* $(Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$*, where* $\Sigma = \Sigma_0 \cup \{\#, @\}$*.* $@ \notin \Sigma_0$ *is a special symbol called the erase symbol. In addition, if* $(a, q) \to (\overline{a}, \overline{q}, d)$ *is an element of transition relation given by* δ*, then* $a = \overline{a}$ *or* $\overline{a} = @$*.*

First of all, we sketch an idea of how a deterministic forgetting automaton (DFA) can store and retrieve information by erasing some symbols on the tape so that the entry picture can be still reconstructed (we follow the description presented in [1]).

Let $A = (Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$ be a DFA. Let $\Sigma_0 = \Sigma \setminus \{@, \#\}$. A performs a computation on a picture O. Let M be the set of tape fields containing O. Let $O(f)$ denote a symbol contained in the field $f \in M$. Then for each $G \subseteq M$ there is $s \in \Sigma_0$ such that $|\{f \in G, O(f) = s\}| \geq \frac{|G|}{|\Sigma_0|}$. Let the automaton A erase fields of G containing the symbol s only. Each such field can therefore store 1 bit of information: the field is either erased or not erased. It is thus ensured that G can hold at least $\frac{|G|}{|\Sigma_0|}$ bits of information. Furthermore, the original symbol of all erased fields in G is known – it is s.

Let us consider M to be splitted into rectangular blocks of the size $k \times l$, where $n \leq k < 2n$, $n \leq l < 2n$ for some n. The minimum value for n will be determined in the following paragraphs. (In the case of just one dimension of the picture being lower than n, the blocks will be only as high – or wide – as the picture. In the case of both dimensions of the picture being lower than n, an automaton processing such a picture can decide whether to accept it or reject it on the basis of enumeration of finitely many cases.)

If both width and height of M are at least n, all blocks contain $n \times n$ fields, except for the blocks neighbouring with the lower boundary of M, which can be higher, and the blocks neighbouring with the right boundary of M, which can be wider. Nevertheless, both dimensions of each block are at most $2n - 1$.

Each block $B_i \subseteq M$ is divided into two parts – F_i and G_i. F_i consists of the first $|\Sigma_0|$ fields of B_i. We can choose the size of the blocks arbitrarily, so a block will always contain at least $|\Sigma_0|$ fields. G_i contains the remaining fields of B_i. Let $s_r \in \Sigma_0$ be a symbol for which $|\{f \in G_i, O(f) = s_r\}| \geq \frac{|G_i|}{|\Sigma_0|}$. The role of F_i is to store s_r: if s_r is the

r-th symbol of Σ_0 then A stores it by erasing the r-th field of F_i. Now A is able to determine s_r, but it needs to store somewhere the information about the symbol originally stored in the erased fields in F_i. A uses the first $|\Sigma_0|$ bits of information that can be stored in G_i. If the erased symbol in F_i was the q-th symbol of Σ_0 then the q-th occurrence of s_r in G_i is erased, allowing A to determine the erased symbol in F_i. This way a maximum of $|\Sigma_0|$ bits of available information storable in G_i will be lost. For any block B_i containing m fields this method allows A to store at least $\frac{m-|\Sigma_0|}{|\Sigma_0|} - |\Sigma_0|$ bits of information in B_i.

In the following text, a region is every rectangular sub-array of tape fields. We can consider such a region to be the picture as well. $\bigoplus[R_{ij}]_{m,n}$ (if defined) is used to denote the region that is the union of R_{ij}'s, where indexes of rows, resp. columns in R_{ij} are less than indexes of rows in $R_{i+1,j}$, resp. columns in $R_{i,j+1}$.

Theorem 3. $\mathcal{L}(CFG) \subset \mathcal{L}(NFA)$

Proof. Let us consider a context-free grammar $G = (V_N, V_T, S_0, \mathcal{P})$, $\mathcal{L}(G) = L$. We describe how to construct a forgetting automaton A that recognizes L. We define Σ as $V_T \cup \{\#, @\}$. Let O be an input picture. The idea of the computation of A is to try to construct a derivation tree T for G such that its root is labeled $(S_0, rows(O) \times cols(O))$ and $p(T) = O$. During the computation, O (more precisely, the region containing the input) will be splitted into disjunct regions, each labeled by an element of $V_T \cup V_N$. We distinguish two kinds of regions: Regions consisting of one field (t-regions) – each labeled by a terminal given by the original content of the field, and regions consisting of more than one field (N-regions) – labeled by a non-terminal. Some of possible regions are *derived.* A derived region is *represented* if there is information determining its position, size and label stored on the tape. We explain later how A derives and represents regions. We consider a bijection between derived regions and vertices of T. Let $m(R)$ denote the vertice corresponding to a region R and $m^{-1}(v)$ the region corresponding to a vertice v.

At the beginning statge, we consider O to be splitted into $rows(O).cols(O)$ t-regions. Each region is derived, represented and corresponds to a leaf of T. These regions are the only derived regions at the beginning. A works in cycles. A cycle includes steps. In a step, A derives a new region. Roughly said, it non-deterministically chooses a (not derived yet) region $R = \bigoplus[R_{ij}]_{s,t}$, where R_{ij} are represented regions or regions derived in the current cycle, R_{ij} labeled A_{ij}, and a production $N \to [A_{ij}]_{s,t} \in \mathcal{P}$ (if such a production does not exists, the computational branch does no accept). R is derived and labeled N. As for the tree T, $m(R)$ is labeled $(N, cols(R) \times rows(R))$ and $p(v(R)) = R$, $m(R_{ij})$ is a descendant of $m(R)$, the edge connecting the vertices is labeled (i, j). A uses the technique of storing information in blocks. We consider O to be divided into rectangular blocks B_i such that $n \leq rows(B_i), cols(B_i) < 2n$, where n is a constant that we derive later. We assume $rows(O), cols(O) \geq n$. The other inputs will be discussed as a special case. Let us describe how to represent regions during the computation. A does not represent any N-region that is a subset of a block – we denote this requirement as (1). We distinguish two types of the remaining N-regions. Let us consider a block B of the size $k \times l$ and a N-region R. Let us denote the four fields neighbouring with the corners of B as the C-fields of B (see Fig. 2). We say that B is a *border* block of R iff $R \cap B \neq \emptyset$ and R does not contain all 4 C-fields of B. A represents a region in its border blocks.

We consider the bits available to store information in B to be organized into groups. Each group has an *usage flag* consisting of two bits determining if the group is *not used* (information has not been stored in the group yet), *used* (stored information is current) or *deleted* (stored information is not relevant anymore). The first state is indicated by two non-erased symbols, the second one by one symbol erased and finally the third one by two erased symbols. One group of bits represents the intersection between a region and a block. Information in a group includes coordinates in the block (one coordinate requires $\lfloor log(2.n) \rfloor$ bits), labels (a non-terminal is represented unary using $|V_N|$ bits) and various "flags" that we describe in the following paragraphs.

Let B be a border block of R. We say that the intersection between R and B is of the *first* type if R contains one or two C-fields of B and of the *second* type if R does not contain any C-field. It is obvious that if R has the intersection of the first, resp. second type with a border block then it has the intersection of the same type with all its border blocks. It means we can denote every N-region having the intersection of the first, resp. second type with its border blocks as N_1-region, resp. N_2-region.

There can be 8 (Fig. 2) different types of the intersection between a N_1-region R and a block B with respect to which C-fields of B are included in R. It means the intersection can be represented using 3 bits determining the type and one or two coordinates. Let us solve the question how many different intersections with N_1-regions A need to represent during the computation in B. B can be a border block of 4 different represented N_1-regions after performing a sequence of cycles. The border

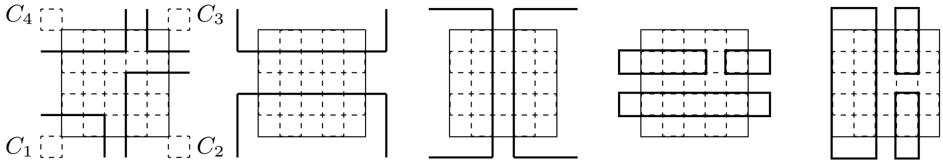

Fig. 2. A block and its C-cells; eight types of intersection between the block and a N_1-region; the horizontal and vertical types of N_2-regions.

(coordinates in B) of one N_1-region can be changed maximally $k + l - 2$ times (before it completly leaves B), because every change increases, resp. decreases at least one of the coordinates. It means it is sufficient if B can represent $8.n$ intersections. Note that more than $8.n$ N_1-regions having the non-empty intersection with the border block B can be represented, however, some of them have the same intersection with B. In addition, if A knows coordinates of R in B it can determine, which group of bits represents R in neighbour blocks. The label of R is represented in one of its border blocks – the correspondent usage flag is of the value "used". If two labels are reserved in each group of bits then there is always at least one not used label in a border block of a new represented region.

We consider N_2-regions to be vertical or horizontal (Fig. 2). Let R be a horizontal, reps. vertical N_2-region. There are three types of intersection between R and B, thus A represents the intersection using tree bits determining if the region is vertical or horizontal and the type of the intersection, two coordinates of the first and the last row of R and, in addition, twice two coordinates with the usage flag that are reserved to represent positions of the leftmost and the rightmost column, resp. the first and the last row of R in correspondent border blocks eventually (the same idea of the representation as in the case of labels). It holds that there is just one border block of R, where one pair of coordinates is marked as used. Let the width of R be $min(cols(R), rows(R))$. We add one more requirement, denoted (2), on the representation of regions: If R is a represented N_2-region then there is not any different represented N_2-region R' of the same width having the same border blocks. Under this assumption, A needs to represent maximally $2.4.2.n = 16.n$ intersections of the second type in a block during the computation (the number of N_2-regions of the width 1 is bounded by $4.max(k, l) \leq 4.2.n$, a N_2-region of a greater width is created as the concatenation of several N_2-regions of less widths, every concatenation decreases the number of represented N_2-region at least by 1).

We complete and summarize what information should be stored in a block during the computation. One bit determines if B is a subset of some derived N-region or not – this information is changed during the computation one time exactly. According to this bit, A determines if a field of a block that is not a border block of any N-region is t-region or not. $8.n$ groups of bits are reserved to represent intersections of the first type. Each group consists of the usage flag, 3 bits determining the type of intersection, two coordinates and two labels, each with the usage flag. Similarly, $16.n$ groups of bits are reserved to represent intersections of the second type. These groups contain one additional information – twice two coordinates and the usage flag. It means we need $O(n.log(n))$ bits per a block, while a block can store $\Omega(n^2)$ bits. It implies that there exists a suitable constant n.

We can describe cycles and steps in more details now. Let d be the maximal number of elemets of the right side of productions in $\mathcal{P}$. In a cycle, A non-determistically chooses a non-represented region R consisting of the set of regions $\mathcal{R} = R_1, \dots R_s$ that are all represented, and a sequence of productions $P_1, \dots P_t$, where $s, t \leq d.4n^2 + 2.4n^2$. A chooses R as follows. It starts with its head placed in the upper left corner of O. It scans row by row from left to right, proceeding from top to bottom and non-deterministically chooses the upper left corner of R (it have to be the upper left corner of an already represented region). Once the corner is chosen, T moves its head to the right and chooses the upper right corner. While moving, when T detects a region R_i first time, it scans its borders and remembers in states the neighbour regions of R_i including their order and the label of R_i as well. When the upper right corner is "claimed", A continues by scanning next rows of R until it chooses the last one. Every time T enters a new represented region (not visited yet), it detects its neighbours. Thanks to mapping of neighbouring relation among R_i's, A is able to move its head from one region to any other desired "mapped" region.

A continues by deriving regions. The first region is derived according to P_1. A chooses $S_1 = \bigoplus[S_{ij}]_{s_1,t_1}$, where each S_{ij} is one one of R_{ij}'s and checks if S_1 can be derived. Let us consider all S_{ij}'s to be deleted from $\mathcal{R}$ and S_1 to be added. A performs the second step on the modified set $\mathcal{R}$ using P_2, etc. In the last step A derives R. All these derivations are performed in states of A only. After that, A records changes on the tape. If the region corresponding to O labeled S_0 is derived then T has been constructed and $O \in L$. On the other hand, let T be a derivation tree for G having its root labeled $(S_0, rows(O) \times cols(O))$ and $p(T) = O$. A can construct T despite the requirements (1) and (2). If R is a region derived using some regions that are subsets of blocks, A derives such regions in one cycle. It requires to choose $d.4n^2$ regions maximally. If R and R' are N_2-regions to be derived having the same border blocks and $R \subseteq R'$, A derives both in one cycle and represents R'. Note that $|R' \setminus R| \leq 2.4n^2$, thus A can choose all needed regions to derive R' as well.

Let us discuss the remaining special case when one of the sizes (e.g. $cols(O) = m$) of O is less than n (if both sizes are less than n then A scans all symbols and accept or reject O immediately). In this case, A needs to represent horizontal N_2-regions in a block only. In addition, maximally $4.m$ different intersections have to be represented in a block (estimated similarly as in the previous case). $O(m.log(n))$ bits are required, while $\Omega(m.n)$ bits can be stored, thus a suitable constant n exists again. □

6 Conclusions

We proved that $\mathcal{L}(CFG)$ does not include all languages in $\mathcal{L}(DFSA)$. It indicates that the presented class is not a suitable candidate for the class of "real" two-dimensional context-free languages. However, in our opinion, this class deserves an attention, because it is based on a natural generalization of CF grammars. We showed how forgetting automata can recognize $\mathcal{L}(CFG)$. The arising question to be study now is whether forgetting automata restricted in some way can characterize it exactly.

References

1. P.Jiřička, V.Král: Deterministic Forgetting Planar Automata, in proceedings of the 4-th Int. Conference: Developments in Language Theory, Aachen, Germany, 1999.
2. D.Giammarresi, A.Restivo: Two-dimensional Languages, In A.Salomaa and G.Rozenberg, editors, Handbook of Formal Languages, volume 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.
3. J.Hopcroft, J.Ullman: Formal Languages and Their Relation to Automata, Addison-Wesley, 1969.
4. A.Rosenfeld: Picture Languages - Formal Models of Picture Recognition, Academic Press, New York, 1979.
5. P.Jančar, F.Mráz, M.Plátek: Characterization of Context-Free Languages by Erasing Automata, Proceedings of MFCS'92, LNCS 629, Springer, 1992, pp. 305–314.
6. P.Jančar, F.Mráz, M.Plátek: Forgetting automata and context-free languages, Acta Informatica 33, Springer-Verlag, 1996, pp. 409–420.
7. M.I.Schlesinger, V.Hlaváč: Deset Kapitol z Teorie Statistickeho a Strukturniho Rozpoznavani, CVUT, Prague, 1999, (in Czech).
8. P.Jiřička: Grammars and automata with two dimensional lists (grids). Master thesis, Faculty of Mathematics and Physics, Charles U., Prague, 1997, (in Czech).